The Report committee for Wenwen Sun

Certifies that this is the approved version of the following report:

Sampling Approaches in Bayesian Computational Statistics with R

Approved By

SUPERVISING COMMITTEE:

Supervisor: _____

Thomas W. Sager

_____

Mary Parker

Sampling Approaches in Bayesian Computational Statistics with R

by

Wenwen Sun, M.A

Report

Presented to the faulty of the Graduate School

To the University of Texas at Austin

In Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Statistics

The University of Texas at Austin

December 2009

Sampling Approaches in Bayesian Computational Statistics with R

By Wenwen Sun, M.S. STAT

The University of Texas at Austin, 2009

Supervisor: Thomas W. Sager

Bayesian analysis is definitely different from the classic statistical methods. Although, both of them use subjective ideas, it is used in the selection of models in the classic statistical methods, rather than as an explicit part in Bayesian models, which allows the combination of subjective ideas with the data collected, update the prior information and improve inferences. Drastic growth of Bayesian applications indicates it becomes more and more popular, because the advent of computational methods (e.g., MCMC) renders sophisticated analysis. In Bayesian framework, the flexibility and generality allows it to cope with very complex problems.

One big obstacle in earlier Bayesian analysis is how to sample from the usually complex posterior distribution. With modern techniques and fast-developed computation capacity, we now have tools to solve this problem.

We discuss Acceptance-Rejection sampling, importance sampling and then the MCMC methods. Metropolis-Hasting algorithm, as a very versatile, efficient and powerful simulation technique to construct a Markov Chain, borrows the idea from the well-known acceptance-rejection sampling to generate candidates that are either accepted or rejected, but then retains the current values when rejection takes place (1). A special case of Metropolis-Hasting algorithm is Gibbs Sampler. When dealing with high dimensional problems, Gibbs Sampler doesn't require a decent proposal distribution. It generates the Markov Chain through univariate conditional probability distribution, which greatly simplifies problems. We illustrate the use of those approaches with examples (with R codes) to provide a thorough review.

Those basic methods have variants to deal with different situations. And they are building blocks for more advanced problems.

This report is not a tutorial for statistics or the software R. The author assumes that readers are familiar with basic statistical concepts and common R statements. If needed, a detailed instruction of R programming can be found in the Comprehensive R Archive Network (CRAN): http://cran.R-project.org

Table of Contents:

Chapter 1 Introduction to Bayesian Analysis with R

1.1Bayesian Analysis Background Overview

The differences between Bayesian statistics and regular (Frequentist) statistics is essentially a different interpretation of what possibility signifies, and thus a different way to make inferences about what a population given that we have a sample of this population.

By Bayesian analysis, we refer to the practical statistical methods that involve the set-up of prior information that usually are based on experience or best guess before experimentation and data collection. Evidence or observations are then used to update or making new inference about the quantities we want to study.

The process of Bayesian analysis can be idealized into the following three steps (2):

1. Setting up a full probability model-- a joint probability distribution for observed and unobserved quantities (e.g., parameters of interest). Model should be consistent with knowledge of the underlying scientific problem or preliminary data analysis.
2. Conditioning on observed data: calculating posterior distribution, which is the conditional distribution of the quantities of research interest, given observed data.
3. Evaluating the fit of the model and the implications of the resulting of posterior distribution: does the model fit the data; is the model robust to the model assumptions (priori distribution choice)?

The second one involves computational methodology and the third one is a delicate combination of techniques and judgment, with proper guidance.

A primary motivation for Bayesian thinking is that it facilitates an easier interpretation of statistical conclusion. For example, a Bayesian interval can be interpreted directly as having a certain probability of containing the unknown quantities of interest, compared with a classic confidence interval, which has to be interpreted with caution as in repeated practice of sampling, from a sequence of similar inference; we have certain probability that the interval might contain the unknown quantities (2).

There has been dramatic growth in the development and application of Bayesian inference in statistics. It is not hard to notice the increase of Bayesian research by the number of published research articles and the number of books. Successful applications of Bayesian data analysis have appeared in many diversified fields, including business, social science, education psychology, economics, epidemiology, geography, imaging, pharmaceutical medicine research, political science, public policy, sports and etc. Thanks to the exponential improvements in modern computers, new computer-intensive methods that generalizing Metropolis-Hasting algorithm and Gibbs sampler, have become possible to compute Bayesian inference for very complex models on large dataset that usually can't be fit by classical statistical models.

1.2 Introduction of the R Environment

To fit Bayesian models, one needs a statistical environment to compute inferences, in which one can define a Bayesian model, use different functions to summarize posterior distribution, simulate samples from the posterior distribution and construct graphs for inference purposes.

R system meets those requirements and provides a wide range of functions for data analysis, calculation and graphic visualization. The R environment is a programming language based on S. Moreover, it is free software with open sources which allows user to extend by adding new functions. Many new developed functions allow user to simplify the programming for certain type of research questions and are provided to other uses in the form of packages that are available in the Comprehensive R Archive Network (CRAN): http://cran.R-project.org. For the special case of Gibbs sampler, the software package Winbugs has allowed non-experts in statistics to fit complex Bayesian models with minimal programming.

1.3 Bayes' Theorem

Bayes' theorem provides a method for inverting conditional probabilities. In its simplest form, if A and B are events and P(B)>0, then

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

For continuous random variables the distribution form of Bayes' Theorem is

$$f_{(X|Y=y)}(x)= f_{(Y|X=x)}(y)f_X(x)/ f_Y(y)= f_{(Y|X=x)}(y)\, f_X(x)/ \int f_{(Y|X=x)}(y)f_X(x)dx$$

For discrete random variables

$$f_{(X|Y=y)}(x)=P(X=x|Y=y)=P(Y=y|X=x)P(X=x)/\sum_x\{P(Y=y|X=x)P(X=x)\}$$

1.4 Bayesian Inference

More generalized, in Bayesian inferential paradigm, the parameters of the probability distributions are usually considered to be fixed but unknown values (some Bayesians might argue that the parameters are random instead of being fixed). The Bayesian approach views the unknown parameters as random variables, with the probability distribution reflecting the analyst's uncertainty about what the true value is. The probability distributions that are associated with the parameters then are used to assign subjective probabilities to the regions of the parameter space to reflect the knowledge of the parameters.

Suppose that Y has a distribution with parameter θ: $f(Y|\theta)$. Let $\pi(\theta)$ represent the density of θ before any data are collected. It is referred as the prior distribution of θ. The prior belief reflects the relative weights that one assigns to the plausible values of the parameter subjectively (3). It may come from previous studies, it may be just purely personal belief, or it may be chosen to have limited influence in the posterior inference (4). Bayesian inference is driven by the likelihood function, usually denoted by $L(\theta|Y)=\prod f(y_i|\theta)$. Having established the prior distribution of θ, Baye's theorem updates prior belief, yielding the conditional density of θ, given observations $y_1, y_2, y_3, \ldots$, which is our posterior density:

$$f_{\theta|Y}(\theta) =\frac{f(y_1,\ldots y_n|\theta)\pi(\theta)}{\int f(y_1,\ldots y_n|\theta)\pi(\theta)d\theta} =\frac{f(y_1,\ldots y_n|\theta)\pi(\theta)}{f(Y)}$$

The posterior distribution summarizes our information about the unknown parameter taking account of the observed data. Then one can compute posterior quantities such as posterior mean, posterior variance and posterior mode for future research.

Note the denominator is the integration of the joint distribution over the parameter, which is the marginal density of the data, independent of parameter θ. So the basic relation is

posterior $\propto$ prior * likelihood.

Posterior distribution is proportional to the product of prior distribution and the likelihood function, up to a constant. A more informative name for the marginal distribution of Y is the prior predictive distribution: prior because it is not conditional on a previous observation of data-generating process; predictive because it is a distribution of an observable quantity (2). It is also a normalizing constant because it ensures the posterior distribution of θ integrates to 1.

f(y)=$\int \pi(\theta) * f(\theta|y) d\theta$

After the data y have been observed, we can predict an unknown observable $\tilde{y}$ in a similar manner. The distribution of $\tilde{y}$ is called the posterior predictive distribution, posterior because it is conditional on the observed y(data vector) and predictive because it is a prediction for an observable $\tilde{y}$:

$$f(\tilde{y}|y) = \int f(\tilde{y}, \theta|y)\, d\theta$$

$$= \int f(\tilde{y}|\theta, y) f(\theta|y)\, d\theta$$

$$= \int f(\tilde{y}|\theta) f(\theta|y)\, d\theta$$

The posterior predictive distribution is displayed as an average of conditional predictions over the posterior distribution of θ. The intermediate step of $f(\tilde{y}|\theta, y) = f(\tilde{y}|\theta)$ follows because $\tilde{y}$ and y are conditionally independent given θ in the model.

The posterior distribution contains a lot of information about the parameter of interest, especially in multi-parameter models. One could just report the entire posterior distribution $f_{\theta|X}(\theta)$ or use graphical tools to display posterior distribution virtually, such as scatter-plot, contour plot (for multi-parameter posterior distribution) (2).

In practice, various numerical summaries are usually desired. For parametric posterior distribution, commonly used location estimators are mean, median and mode(s) of the distribution; variability/spread/scale estimators are standard deviation, IQR (inter-quartile range), coefficient of variance, etc. Mean measures the average value of the variable. Mode is interpreted as the value that occurs most frequently in a data set or a probability distribution (note mode is not necessarily unique, since the same maximum frequency may be attained at different values). Mode is important in computation statistics especially for more complex problems, as it is computationally easier.

Besides those point estimators, one could also construct interval estimate. In general, intervals are constructed by stimulation from posterior distribution. Just like the way to construct confidence interval, 100(1-α)% central interval corresponds to the range of values above and below the regions which lie exactly 100(1/2α)% of posterior probability. A slightly different but more useful method for multimodal distribution is to compute a region of the highest probability density (HPD): a region of values that contains 100(1-α)% of the posterior probability and also

has the property that the density within the region is never lower than the outside (2). It is available in R through the “HPDregionplot” command.

Chapter 2 Prior Distribution Selection

2.1 Informative Prior Distribution

A first step in Bayesian Data Analysis is the choice of a proper prior distribution for the parameter (vector) of interest. Usually the choice of prior is a case by case scenario based on how much information you have at hand and your subjective belief about the parameter. Here, we categorize two different prior distributions: Informative prior distribution and Non-informative prior distribution.

2.1.1 Discrete Prior

Informative prior refers to a prior that reflects substantial information about the parameters of interest. For example, in a simple binomial model, we want to estimate the proportion of the results from a sequence of "Bernoulli Trails", x_1, x_2, x_3,…. x_n, which can only be success (normally 1) or failure (0). Because of the exchangeability of the n independent trials, the results can be summarized by the total number of successes out of the n trials, which we denote $y=\sum_1^n x_i$. The binomial sample model states:

$p(y|n,p)=\text{Binomial}(y|n,p)=\binom{n}{y} p^y (1-p)^{n-y}$

In a simple application of this binomial model, we want to study the proportion of a thumbtack (I didn't choose the traditional coin toss because we all expect to see about 50% tosses to be head if the coin is fair; while for thumbtack, we have no expectation) landing completely on the flat metal head (considered as "success") as opposed to landing tilted on the tail (considered as "failure"). One way to assess the prior for this proportion is to write down a list of plausible values of p and then assign the frequency/weight to those values. In our study, we believe that

0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95

Are the possible values for p. Based on the belief, we assign the following values to the corresponding weights:

1, 1, 3, 3, 5, 5, 3, 3, 1, 1

Those weights can be easily converted to prior probability by dividing each weight by the sum of the total weights. Then in our experiment, we observe 7 heads and 3 tails. In R, the codes are straightforward:

```
# Discrete prior

p = seq(0.05, 0.95, by = 0.10)

prior = c(1, 2, 2, 3, 7, 7, 4, 3, 1, 1)

prior = prior/sum(prior)

plot(p, prior,type = "h", ylab="Prior Distribution")

data=c(7,3)
```

```
post=pdisc(p, prior, data)

cbind(p, prior, post)

plot(p, post, type = "h", ylab="Posterior Distribution")
```

The output is

```
   p      prior         post
 [1,] 0.05 0.03225806 2.197559e-08
 [2,] 0.15 0.06451613 6.885022e-05
 [3,] 0.25 0.06451613 1.689560e-03
 [4,] 0.35 0.09677419 1.739071e-02
 [5,] 0.45 0.22580645 1.427754e-01
 [6,] 0.55 0.22580645 3.186061e-01
 [7,] 0.65 0.12903226 2.758274e-01
 [8,] 0.75 0.09677419 2.052815e-01
 [9,] 0.85 0.03225806 3.549654e-02
[10,] 0.95 0.03225806 2.863881e-03
```

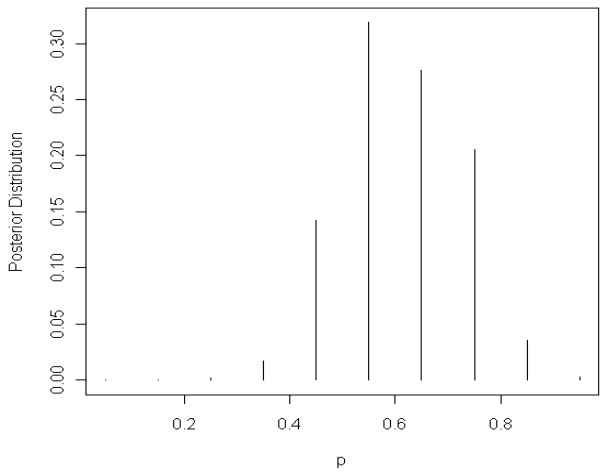


Figure1. Posterior distribution using a discrete prior distribution

Here we note the most of the posterior probability is concentrated on values of p=0.55 and p=0.65. Furthermore, we notice, if we combine the probabilities for the three most likely values, we can say the Probability that proportion p falls in the set {0.55, 0.65, 0.75} is equal to 0.800.

2.1.2 Conjugate prior

Continuing with the binomial model, since proportion is continuous, instead of assigning a discrete point prior, we can construct a continuous prior that reflects the initial belief. A convenient family of densities for proportion is the beta distribution with density proportion to $g(p) \propto p^{\alpha-1}(1-p)^{\beta-1}$, $0<p<1$

Beta distribution is characterized by two parameters α and β, which means we can specify a prior beta distribution by assigning proper values to α and β. Beta distribution is very diversified, with different values of α and β, it shows different shapes as follows.

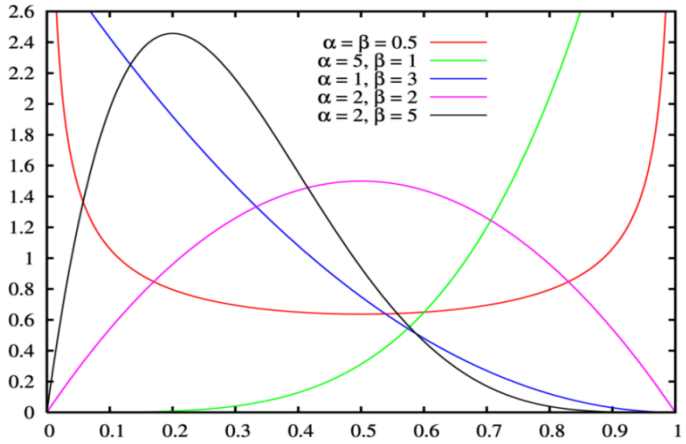


Figure2. Beta distributions with different parameters (from Wikipedia).

With more information regarding the data, one can easily match a proper set of parameters α and β to reflect personal belief. We also notice, our likelihood function here has the form of $L(y|p)=\prod p(y_i|p) \propto p^s(1-p)^f$, where s stands for number of successes and f stands for number of failures. The prior beta distribution is of the same form. Combining the prior with the likelihood function, we can easily show that the posterior distribution is also of the beta form, with parameter α+s-1 and β+f-1.

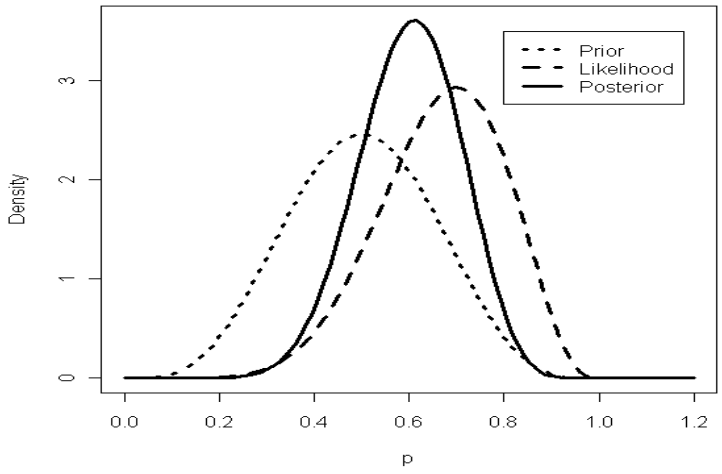
$g(p|y) \propto p^{\alpha+s-1}(1-p)^{\beta+f-1}$, $0<p<1$.

The calculation is very easy to program in R as followed:

```
p= seq(0, 1, length = 100)

a = 5

b = 5

s = 7

 f = 3

prior = dbeta(p, a, b)

like = dbeta(p, s+1, f+1)

post=dbeta(p, a+s,b+f)
```

```
curve(dbeta(x,a+s,b+f), from=0, to=1,
+       xlab="p",ylab="Density",lty=1,lwd=4)
curve(dbeta(x,s+1,f+1),add=TRUE,lty=2,lwd=4)
curve(dbeta(x,a,b),add=TRUE,lty=3,lwd=4)
legend(.7,4,c("Prior","Likelihood","Posterior"),
+     lty=c(3,2,1),lwd=c(3,3,3))
```

The output is the following figure:



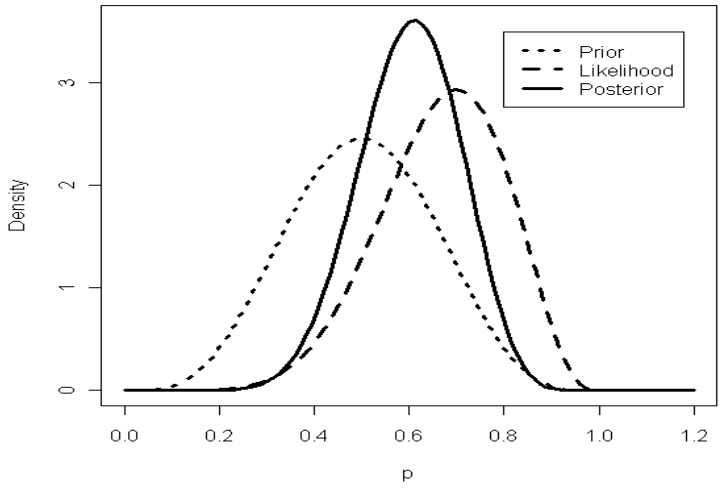


Figure3. Prior, likelihood and posterior distribution using beta conjugate prior.

In Bayesian probability theory, a class of prior probability distributions $p(\theta)$ is said to be conjugate to a class of likelihood function $p(x|\theta)$ if the resulting posterior distributions $p(\theta|x)$ are in the same parametric family as $p(\theta)$; the prior and posterior are then called conjugate distributions, and the prior is called a conjugate prior for the likelihood. For example, the beta prior distribution is a conjugate family for the binomial likelihood (2). More generally speaking, the exponential families are the only classes of distributions that have natural conjugate prior distribution. As we can tell from the beta prior distribution, a beta (1,1) is actually the uniform distribution. The "principle of insufficient reason" claims that if nothing is known for the parameter, then a uniform specification is appropriate (2).

Conjugate prior distribution definitely has practical advantages in that not only it simplifies the computation of the posterior distribution with a known analytical form, but also they can be useful later as building blocks for more complex models, such as multi-level hierarchical models.

2.1.3 Histogram prior

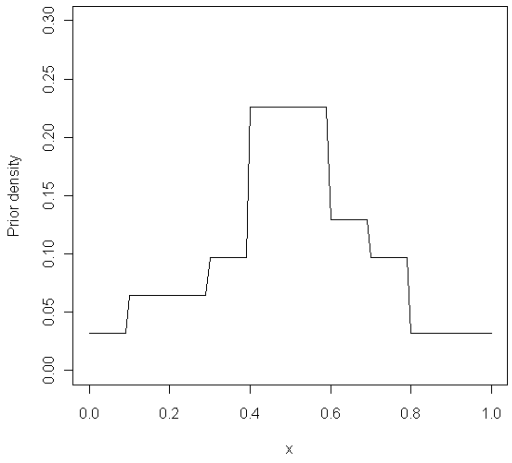
Typically, an ideal prior distribution should contains all plausible values of the parameter, but the distribution need not be concentrated around the true values, because after obtaining the information from data, which will often outweigh prior specification and become dominant in the posterior calculation. When a conjugate prior is not reasonable, we have to choose other more realistic prior distributions. Although a non-conjugate prior can make interpretation of posterior distribution less transparent and more difficult for computation, it doesn't pose any new conceptual problems. It is still possible to perform posterior computation with non-conjugate arbitrary prior by a "brute-force" method (5), as discussed in the next paragraph.

First, choose a grid of values of p over an interval within the posterior density. Compute the product of the prior g(p) and the likelihood function L(y|p) on that grid. Posterior distribution can be approximated by a discrete distribution on the grid, which is obtained through normalizing each product by dividing the sum of those products. In R, the samples can be taken from this posterior distribution by a simple command sample. We continue with the binomial model about the proportion of flipping thumbtacks. The codes are as followed:

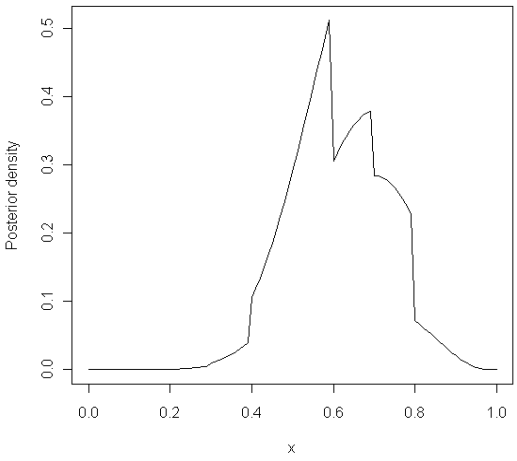
```
#  HISTOGRAM PRIOR

midpt = seq(0.05, 0.95, by = 0.1)

prior = c(1, 2, 2, 3, 7, 7, 4, 3, 1, 1)

prior = prior/sum(prior)

curve(histprior(x,midpt,prior), from=0, to=1, ylab="Prior density",ylim=c(0,.3))

# The curve command chooses an appropriate number of values

# to form a grid between the end points and computes the

# value of the posterior distribution at each point of that grid.

s = 7

f = 3
```

```
curve(histprior(x,midpt,prior)*dbeta(x,s+1,f+1),  from=0, to=1, ylab="Posterior density")
x=seq(0,1,length=500)
post=histprior(x,midpt,prior) * dbeta(x,s+1,f+1)
post=post/sum(post)
ps=sample(x, replace=TRUE,prob=post)
hist(ps)
```

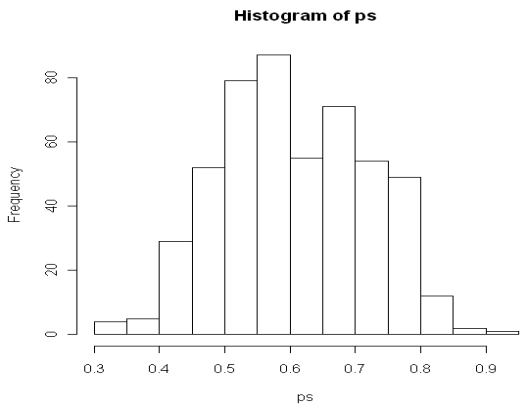
The output are the following figures:



(a)



(b)



(c)

Figure4. (a) Prior distribution by approximating over discrete grids. (b)Posterior distribution from the histogram prior. (c) Histogram of the posterior distribution.

By the brute force method, we compute the posterior distribution over the values of the grid, by multiplying the histogram prior with the likelihood functions. To obtain a random sample from the posterior we calculated, we normalize the product by dividing the sum of all products over that grid. The simulated samples can be summarized in a histogram, which can be used later to do inference on statistics of interest.

2.2 Non-informative Prior

Some Bayesians say that if you do not have sharp ideas about the parameters, then you don't know very much about your problem. While, in practice, it often occurs that one may have incomplete information about the parameter, which would result that more than one distribution would match with the given information. It is always desirable that inferences from the posterior distribution not be dependent heavily on the exact form of the prior. A Bayesian analysis model is said to be robust to the prior if the inference is insensitive to prior. In an even worse situation, where prior information is so poor, such as no population basis at all, it is still important to choose the right prior that plays minimal roles in the posterior distribution.

Strategies are available to reduce the sensitivity. One simple way is to use a prior that are rather dispersed and flat over the support of a broader region than the data (4). An more informal approach is to use Jeffrey's principle that defines the non-informative prior density as g(θ) $\propto$[I(θ)]$^{1/2}$, where I(θ) is the Fisher Information for parameter θ:

$$I(\theta)=E[(\frac{dlogp\ (y|\theta)}{d\theta})^2|\theta]=-E[\frac{d^2 logp\ (y|\theta)}{d\theta^2}|\theta] \quad (6)$$

Consider the binomial distribution, y~binomial(n,p). The log likelihood function is

$$Logp(y|\theta)=log[\binom{n}{y} p\theta^y (1-\theta)^{n-y}]$$

$$= log\binom{n}{y} + y*log(\theta)+(n-y)*log(1-\theta)$$

We then evaluate Fisher's information:

$I(\theta) = -E[\frac{d^2 logp\,(y|\theta)}{d\theta^2}|\theta] = \frac{n}{\theta(1-\theta)}$

By Jeffrey's principle, the prior density is then $g(\theta) \propto \theta^{-1/2}(1-\theta)^{-1/2}$, which correspond to a beta(1/2, 1/2) distribution. Here we actually encounter a problem, when we try to find prior density for parameter θ. In our example, θ is a proportion, which means $0<\theta<1$. It is always helpful to re-parameterize such parameters, so that they can be real-valued with support $(-\infty, +\infty)$. For example, if we have a positive parameter, such as a variance, we can always re-parameterize by logarithm transformation. And for a proportion with support (0, 1), such as the one we have, we can do the logit transformation of taking $\log(\frac{\theta}{1-\theta})$. With such a transformation, if we denote $\delta = \log(\frac{\theta}{1-\theta})$, then $g(\text{logit}(\theta)) = f(\delta) = g(\frac{e^{\delta}}{1+e^{\delta}}) * \frac{d\theta}{d\delta} = n$, which is a constant. So after transformation, we get a prior $\propto$ constant, corresponding to a beta(1,1), equivalent to a uniform prior, just the same as what the "principle of insufficient reason" claims: when no information is available, a uniform is appropriate (2).

Fisher's information can be extended to multi-parameter models. If θ is a parameter vector, $I(\theta) = E\{I'(\theta)I'(\theta)^T\} = -E\{I''(\theta)\}$, where $I''(\theta)$ denotes the pxp matrix having (i, j)th element given by $\frac{d^2 I(\theta)}{d\theta_i \theta_j}$. $I(\theta)$ may sometimes be called the expected Fisher information to distinguish itself from $-I''(\theta)$, which is the observed Fisher information(4). One problem may occur that you might end up with an improper prior, which means the chosen prior may not sum up to 1 or to a finite number. We can always solve it by finding a constant to re-normalize it or keep it in mind, and find that constant in our posterior calculation later on.

Chapter 3 General Sampling Strategy from Posterior Distribution

3.1 Direct Simulation

After choosing the appropriate prior distribution, the following calculation to get the posterior distribution seems straightforward conceptually. In simple non-hierarchical Bayesian models, it is often easy to draw directly from a posterior distribution that is one of many common and simple forms, especially if a conjugate prior distribution has been assumed. However, nearly all Bayesian posterior distributions are not members of standard parametric families. In addition, there can be more difficulties besides the absence of an obvious method to sample from target distribution. For example, in a Bayesian analyses, the posterior distribution may be known only up to a multiplicative constant. In such case, if we can find a method to sample the target distribution, it can only be evaluated up to that constant. Fortunately, various techniques are available to simulate directly from a target density in this setting.

3.1.1 Inverse cumulative distribution function

For any continuous distribution function F, if $U \sim Unif(0,1)$, then $x=F^{-1}(U)=\inf\{x:F(x)\geq U\}$, where F is the cumulative distribution function of x. One prerequisite is that F^{-1} does exist and is easy to compute. If F^{-1} is not available, but F is available, a crude approach can be built by approximating $u_i=F(x_i)$ with calculation at each grid of points. Then linearly interpolate between the two nearest grid points for which $u_i<U<u_j$ by

$$X=\frac{u_i-U}{u_j-u_i}x_i+\frac{U-u_i}{u_j-u_i}x_j \quad (4)$$

This method can be generalized to high dimensions, but the complexity of computation is daunting.

3.1.2 Rejection Sampling

If the target density is the posterior distribution $p(\theta|y)$, which is really complex but can be calculated, at least to a constant, then we can use a general-purpose algorithm called rejection sampling to obtain random draws directly from the target distribution. This approach relies on a proposal density, let's denote it as g(.), which has to satisfy the following conditions (5):

1. It is easier to sample from g.
2. The proposal density g resembles the target density p (here is the posterior density) in terms of location and scale.
3. For all possible θ's, there exists a constant c, such that $p(\theta|y)\leq cg(\theta)$.

With the following examples, we will discuss the procedure of rejection sampling and how to choose a proper proposal density.

Supposed we found such a density g(.) with these properties, rejection sampling proceeds as follows:

1. Simulate θ from g(.).
2. Simulate U from Unif(0,1)
3. If $U\leq p(\theta|y)/cg(\theta)$, accept θ as a random sample from the target density $p(\theta|y)$, otherwise, reject θ and return to step 1 to start over.
4. Continue these steps until one has collected a sample of desired size.

Note, in rejection sampling, there is no approximating involved. To see that the accepted samples actually have the same distribution of the target density, we can prove by Baye's Theorem as follows:

p(accepted|θ)=p(U≤p(θ|y)/cg(θ))= p(θ|y)/cg(θ)

The last equality follows simply because U is uniform on (0, 1). Therefore, the total probability of acceptance for any iteration is:

$$\int p(\text{accepted}|\theta)g(\theta)d\theta = \int \left[\frac{p(\theta|y)}{cg(\theta)}\right] g(\theta)d\theta = 1/c$$

So, for every θ,

probability(θ|accepted) $= \frac{p(\text{accepted }|\theta)g(\theta)}{\int p(\text{accepted }|\theta)g(\theta)d\theta} = \frac{\left[\frac{p(\theta|y)}{cg(\theta)}\right]g(\theta)}{1/c} =$ p(θ|y), which is exactly our target posterior density.

In Bayesian setting, frequently, the target posterior density is un-normalized. Let's denote q(θ|y)=α*p(θ|y), where α is the unknown normalizing constant, equal to $\int$ p(θ|y) *L(y_1,y_2,….y_n|θ)dθ. Rejection sampling can be applied in such cases as well. Suppose we have a proposal density g(.). For every simulated θ from (g.), we will accept θ as a draw from the target q(θ|y), if U≤q(θ|y)/cg(θ), and reject θ otherwise. The only difference here is we have q(θ|y) instead of p(θ|y). The sampling distribution remains unchanged because the unknown α will cancel out in the numerator and the denominator. The proportion of kept draws is 1/αc.

Rejection sampling is one of the most useful methods for simulating samples from a variety of target distribution. The main obstacle is to search for a suitable proposal density g and the constant value c. The probability of acceptance for an individual draw is p(θ|y)/cg(θ). And the proportion of accepted draws is 1/c. One can monitor the algorithm by choosing the right constant. An efficient proposal density should generate a high acceptance rate. It is not hard to see that a good proposal density should be proportional to p(θ|y). Ideally, g(.) should be chosen to be proportional to p(θ|y), so that with a suitable c, we can accept every single draw with probability 1. If g(.) is not proportional to p(θ|y), c has be large so that cg(.) can cover the target density over the entire support of the parameter θ. Inevitably, we will have some regions that have high probability of acceptance versus some regions that we have low acceptance. Overall, the acceptance rate is not high, and such choice of proposal density and constant c are not very efficient.

Multivariate targets can also be sampled by rejection sampling, provided a suitable multivariate proposal density is available. It doesn't impose any conceptual problems.

The following is a simple example of R codes for rejection sampling from a beta distribution: x~beta(3,4), which can be written as p(x)=)= 60*(x^3)*((1-x)^2).

```
betapdf=function(x){

+ 60*(x^3)*((1-x)^2)}

curve(betapdf,0,1)

abline(2.1,0)
```

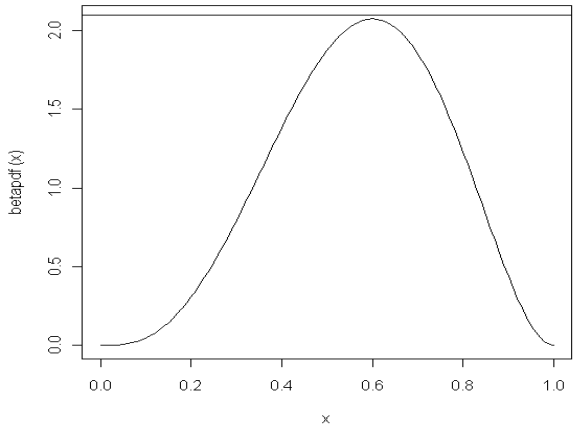


Figure5. Beta(3,4) distribution with a uniform proposal function.

By plotting the density function, we found we can use a unif(0,1) as our proposal density function g(.) with a constant c=2.1, the product of which will cover the target density over the support. Then, we want to using the Acceptance-Rejection method to see how many runs are needed to generate 1000 Beta(4,3) random variables.

```
n <- 1000

k <- 0      #counter for accepted

j <- 0      #iterations

z <- numeric(n)

while (k < n) {

+       u <- runif(1)

+       j <- j + 1

+       y <- runif(1)  #random variate from g

+       if (60/(2.1)*(y^3) * ((1-y)^2) > u) {

+          #we accept y

+          k <- k + 1

+          z[k] <- y

+       }

+    }

j

[1] 2093
```

hist(z, prob=TRUE, breaks=50)

curve(betapdf,0,1,add=TRUE)

So, we need 2093 iterations to generate a 1000 samples, the acceptance rate is 1000/2093≈0.4778. We also construct a true target density function beta(3,4) compared with the histogram of samples we got from rejection sampling.

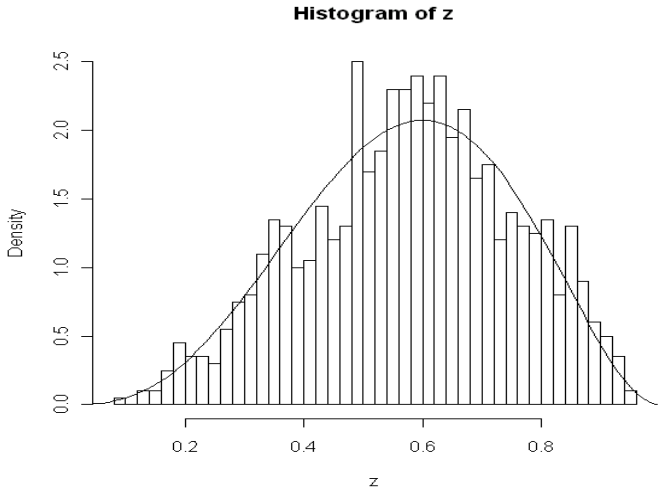


Figure6. Histogram of the posterior simulation with rejection sampling with the theoretical beta(3,4) distribution overlay.

3.2 Monte Carlo Integration

In Bayesian analysis, after sample from the posterior distribution, we need to summarize a posterior distribution with statistics or any function of parameter of interest that is where the original Monte Carlo integration comes in handy. It was originally developed by physicists to use random generated numbers for calculating integrals (6). The Monte Carlo approach is very convenient for the posterior distribution, which we can sample directly from.

Suppose that θ has a posterior distribution p(θ|y) and we are interested in learning about a particular function of the parameter h(θ). The mean of h(θ) is given by

$$E(h(\theta)|y)=\int h(\theta)p(\theta|y)d\,\theta$$

First, we need to simulate random draws θ_1, θ_2, θ_3,….θ_n from the posterior distribution p(.). Then the Monte Carlo estimate of the mean of h(θ)|y is given by the sample mean:

$$\bar{h}=\frac{\sum_{j=1}^{m} h(\theta^j)}{m}$$

The simulation standard error of this estimate is given by

$$se_{\bar{h}}=\sqrt{\frac{\sum_{j=1}^{m}(h(\theta^j)-\bar{h})\text{^}2}{m(m-1)}}$$

Monte Carlo Integration applies majorly to multivariate distribution, where the direct integration is too complicated or computational-expensive.

3.3 Importance Sampling

A little twist occurs often in Bayesian statistics. In many situations, the normalizing constant is unknown and would be hard to calculate. So the evaluation of E(h(θ|y)) will be a ratio:

E(h(θ)|y)=$\frac{\int h(\theta)q(\theta|y)d\theta}{\int q(\theta|y)d\theta}$, where q(θ|y) is the un-normalized posterior distribution.

If sampling directly from q(θ|y) is difficult, we might use an alternative approach called importance sampling, given that we can construct a probability density g(.) that we can simulate easily and that approximates the non-normalized posterior distribution q(.). We can rewrite the mean of h(θ|y):

E(h(θ)|y)=$\frac{\int h(\theta)\left(\frac{q(\theta|y)}{g(\theta)}\right)g(\theta)d\theta}{\int\left(\frac{q(\theta|y)}{g(\theta)}\right)g(\theta)d\theta}$

=$\frac{\int h(\theta)w(\theta)g(\theta)d\theta}{\int w(\theta)g(\theta)d\theta}$, where w(θ)=$\frac{q(\theta|y)}{g(\theta)}$, called the weight function (5).

We simulate samples from g(.), then the importance sampling estimate based on the Monte Carlo approach is

$\bar{h}_{IS}=\frac{\sum_{j=1}^{m} h(\theta^j)w(\theta^j)}{\sum_{j=1}^{m} w(\theta^j)}$

This is the importance sampling estimate. The simulation standard error of this importance sampling estimate is given as:

$se_{\overline{hIS}}=\sqrt{\frac{\sum_{j=1}^{m}((h(\theta^j)-\bar{h})\mathrm{w}(\theta^j))\text{^}2}{\sum_{j=1}^{m}\mathrm{w}(\theta^j)}}$

Both estimators converge by the law of large number as $n\to\infty$, as long as the importance sampling function g(.), also called the envelope function, has the support at least as large as that of the target q(.). A good estimate needs to be precise, in other words, with small variance. In order to have small variance of our importance sampling estimate, a proper proposal function g(.) has to satisfy those conditions: it should be easy to simulate samples from, such as of a familiar function form. Also, it should mimic the posterior density p(.) and p(.)/g(.) has to be bounded(4). One can always monitor the choice of g(.) by inspecting the values of the simulated weights w(θ^j). We can compute a histogram of the simulated weights and if there are not any unusually large weights, then it is likely that the weight function is bounded. Or, for different choice of proposal function, just check the variance of the estimate. I include an example of R code for comparing 2 different proposal functions by importance sampling.

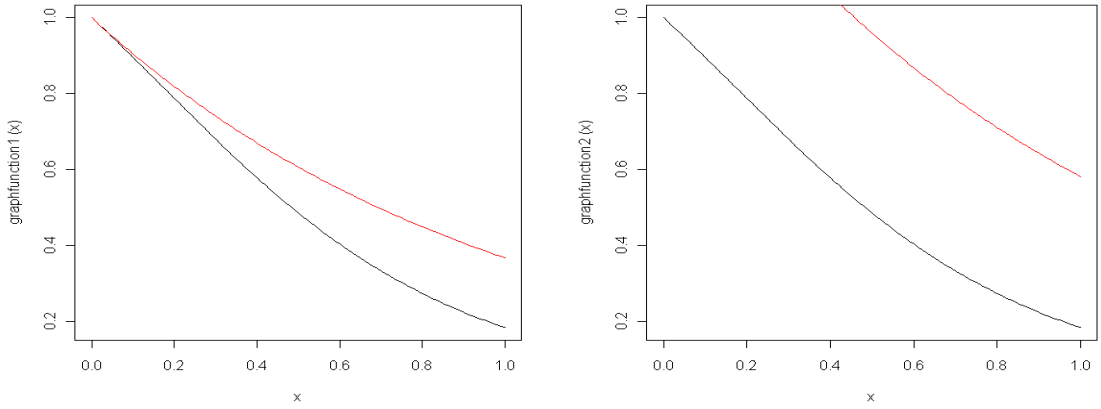
I want to estimate$\int_0^1 \frac{e^{-x}}{1+x^2}dx$. The possible proposal densities are $g_1(x)=e^{-x}$ and $g_2(x)=e^{-x}/(1-e^{-1})$.

```
n <- 10000

theta.hat <- variance<- st.dev <- numeric(2)

g <- function(x) {

+ exp(-x)/(1+x^2) * (x > 0) * (x < 1)
```

```
+ }
#1 st candidate of g(x)
x<-rexp(n)
f<-g(x)/(exp(-x))
theta.hat[1]=mean(f)
theta.hat[1]
[1] 0.5288185
variance[1]=var(f)
variance[1]
[1] 0.1748755
st.dev[1]=sd(f)
st.dev[1]
[1] 0.4181811

#2nd candidate of g(x)
u<-runif(n)
x<-log(1/(1-u*(1-exp(-1))))
f<-g(x)*exp(x)*(1-exp(-1))
theta.hat[2]=mean(f)
theta.hat[2]
[1] 0.5259794
variance[2]=var(f)
variance[2]
[1] 0.009230845
st.dev[2]=sd(f)
st.dev[2]
[1] 0.09607729
rbind(theta.hat, st.dev)
                [,1]          [,2]
```

theta.hat 0.5288185 0.52597941

st.dev 0.4181811 0.09607729



(a) (b)

Figure7. Target distribution (black) with different proposal distributions (red): (a) proposal distribution $g_1(x)$; (b) proposal distribution $g_2(x)$.

Clearly from the two graphs above, we see $g_2(x)$ is closer to the target function in terms of shape than $g_1(x)$. Therefore, from the variance output, we see the importance sampling estimate using $g_2(x)$ has a much smaller standard error (0.09607729) compared with the standard error (0.4181811) using $g_1(x)$.

Chapter 4 Markov Chain Monte Carlo (MCMC)

4.1 Markov Chain Properties

The previously discussed general methods for simulating from an arbitrary target distribution are very useful, but can be difficult to set up, since rejection sampling as well as importance sampling both requires the construction of a suitable proposal density. It may be difficult to find such a proposal density for high-dimensional problems. An alternative is the class of Markov Chain Monte Carlo (MCMC) methods, including Metropolis-Hasting algorithm and Gibbs sampler, which have become more and more popular. Indeed, it can be fairly said that MCMC methods largely account for the renaissance of Bayesian methods in the last 20 years. The so-called Markov Chain Monte Carlo approaches use the previous values to randomly generate the next sample values, forming a Markov Chain (A Markov Chain is a stochastic process in which the probability of moving from the current state to the next state depends only on the current state).

MCMC methods have their roots in Metropolis algorithm, which was originally proposed by physics (7) to compute complex integrals by expressing them as expectation for some distributions and then estimate with samples draw from the distributions. For complex problems, it may be impossible to implement a method for generating independent observations from the target density p(.), using the general-purpose sampling methods such as rejection sampling, importance sampling and etc, that's where the first "MC" comes to rescue. Markov Chain simulation draws samples from approximate distributions and corrects those draws to better approximate the target distribution. The samples are drawn sequentially, with the current drawing value depending on the last value drawn; hence, they form a Markov Chain (2). The Markov property plays an important role in the success of this method as the approximate distribution is improved at each step of iteration in the simulation, in the sense of converging to the target distribution.

Several properties of Markov Chain are very important to understand the MCMC methods and the following Metropolis-Hasting algorithm. Let's denote X_t to be the value of a random variable at time t, with support of the range of possible X values. The Markov process is defined if the transition probabilities between different values in the state space depend only on the random variable's current state, i.e:

$$Pr(X_{t+1}=s_j|X_0=s_0, X_1=s_1, \ldots X_t=s_i)=Pr(X_{t+1}=s_j|X_t=s_i)$$

So for a random variable from Markov process, the only useful information to predict the next value is the current value. Knowledge of the all the earlier states does not change the transition probability. Markov chain refers to a sequence $\{X_1, X_2, X_3 \ldots X_t\}$ generated by this Markov process. Usually, we describe movement between states in terms of transition probabilities, sometimes called transition kernel, which shows the likelihoods of moving between all possible states in a single step in a Markov Chain.

$$P(i,j)=P(i\rightarrow j)= Pr(X_{t+1}=s_j|X_t=s_i)$$

For any given time t, we normally use $\pi_j(t)=Pr(X_t=s_j)$ to denote the current state of x at time t. The chain is usually constructed by specifying an initial vector (0) (usually all elements in (0) are zero except for one single element to be 1, corresponding the starting from a particular state).

If we define the probability transition matrix P, then we have

$$\pi(t+1)=\pi(t)P$$

Using the matrix form, we can immediately see how to iterate the equation, as

$\pi(t)= \pi(t-1)P=(\pi(t-2)P)P=(t-2)P^2$

Likewise, we have

$\pi(t)= \pi(0)P^t$

There, if we define an n-step transition probability $p_{ij}^{(n)}$ as the probability that the random variable is at state j at step n given it starts at state i at n-step ago.

$p_{ij}^{(n)}=Pr(X_{t+n}=s_j|X_t=s_i)$

And from linear algebra, we know that $p_{ij}^{(n)}$ is the ij-th element of P^n.

If there exists a positive integer such that $p_{ij}^{(n_{ij})}>0$ for all i, j, which means it is possible to move from every state to every state in one or more steps, the Markov Chain is said to be irreducible (5). Given that currently X is in a certain state, if it has to take certain regular intervals to return to the same state, this Markov Chain is said to be periodic, otherwise, it is aperiodic (5). It is clear that the initial values definitely have effect on the state where the random variables are with such a Markov process. But after large number of the iteration, the effect from the initial values wears out; a Markov Chain converges to a stationary distribution. For Markov Chain that is irreducible and aperiodic, there is a unique stationary distribution. A Markov Chain may reach a stationary distribution π^*, which satisfies

$\pi^* = \pi^*P$

In other words, π^* is the left eigenvector associated with the eigenvalue $\lambda=1$ of P (10). The limiting distribution of Markov Chain, as the number of steps approaches infinity, will be equal to the stationary distribution. Markov Chain is usually constructed so that the stationary distribution is our target distribution. One possible method to determine the stationary distribution for transition matrices is to raise the transition matrix to a large power (5). We illustrate the R code with a discrete Markov Chain.

```
T=matrix(c(.2,.8,0,0,0,.2,.2,.6,0,0,0,.4,.2,.4,0,0,0,.6,.2,.2,0,0,0,.8,.2),
+ byrow=TRUE,nrow=5,ncol=5)
T
     [,1] [,2] [,3] [,4] [,5]
[1,]  0.2  0.8  0.0  0.0  0.0
[2,]  0.2  0.2  0.6  0.0  0.0
[3,]  0.0  0.4  0.2  0.4  0.0
[4,]  0.0  0.0  0.6  0.2  0.2
[5,]  0.0  0.0  0.0  0.8  0.2
s=array(0,c(100000,1))
```

```
s[1]=1
for (j in 2:100000)
+ s[j]=sample(1:5,size=1,prob=T[s[j-1],])
m=c(5000,10000,50000,100000)
for (i in 1:4)
+ print(table(s[1:m[i]])/m[i])
```

The output is the following:

```
   1     2     3     4     5
0.0656 0.2576 0.3730 0.2416 0.0622

   1     2     3     4     5
0.0640 0.2514 0.3766 0.2451 0.0629

    1      2      3      4      5
0.06456 0.25456 0.37654 0.24344 0.06090

    1      2      3      4      5
0.06413 0.25107 0.37436 0.24865 0.06179
```

Here we summarize the frequency of visits to the five different states after 5000, 10000, 50000, 100000 steps of the chain, and convert the counts to relative frequency by dividing by the number of steps. It seems from the output that the relative frequencies of the states converge to the stationary distribution π^*=(.062,.25,.376,.25,.062). It can be confirmed that π^* is indeed the stationary distribution of this discrete chain by multiplying π^* by the transition matrix T:

```
w=matrix(c(.062,.25,.376,.25,.062),nrow=1,ncol=5)
w%*%T
      [,1] [,2]   [,3] [,4]   [,5]
[1,] 0.0624 0.25 0.3752 0.25 0.0624
```

4.2 Metropolis-Hasting Algorithm

4.2.1 General Metropolis-Hasting Algorithm

The MCMC strategy is to set up an irreducible, aperiodic Markov Chain for which the stationary distribution equals the posterior distribution of interest. A general method of constructing a Markov Chain is by a Metropolis-Hasting Algorithm including special cases of the independence chain, random walk chain and the Gibbs Sampler. The algorithm has to specify, for a given state X_t, how to generate the next value X_{t+1}. In general M-H algorithm, a candidate point Y has to be generated from a proposal distribution. If Y is accepted with some modified acceptance probability, the chain will move to state Y at time t+1 and $X_{t+1}=Y$. Otherwise, the chain stays at its current value with $X_{t+1}=X_t$. Also, note that the proposal distribution can be dependent on the current value X_t (3).

The Metropolis-Hasting Sampler generates a Markov Chain with target distribution f as follows:

1. Start with any initial value X_0 which is in the support of the target distribution p(.) and $p(X_0)>0$

2. Using the current value X (X_0 for the first draw), sample a candidate Y from a jumping distribution $q(X_t, Y)$, which is the probability of returning a value of Y given a previous value of X_t. This distribution is also referred to as the proposal or candidate-generating distribution.

3. Generate an independent U from uniform (0,1).

4. If $U \leq \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)}$, accept Y and set $X_{t+1}=Y$; otherwise, set $X_{t+1}=X_t$.

5. Return to step 2 and increment of t.

We can see that in step (4), the candidate point Y is accepted with probability

$$\alpha(X_t, Y)=\min(1, \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)})$$

We can see since the acceptance probability is expressed as a ratio, suppose our goal is to draw sample from some distribution p(.) with p(.)=f(.)/K, where the normalizing constant K is unknown and difficult to compute, this algorithm still works in the situation. Hence, it is a very popular approach in Bayesian analysis where the K is always unknown and hard to compute.

To demonstrate that the M-H sampling generates a Markov chain with stationary density is the target density p(x), the sufficient condition is to show the transition kernel satisfies the balance equation, for all i and j:

$$p(i,j)\pi_i^* = p(j,i)\pi_j^*$$

It is also called the reversibility condition (1,7). Under the M-H algorithm, we sample from $q(X_t,Y)=Pr(X_t \rightarrow Y|q)$ and accept the move with probability $\alpha(X_t, Y)$, so the transition kernel is

$$Pr(X_t \rightarrow Y)= q(X_t,Y)\ \alpha(X_t, Y)= q(X_t,Y)\ \min(1, \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)})$$

If this transition kernel satisfies

$$q(X_t,Y)\ \alpha(X_t, Y)p(X_t)= q(Y,X_t)\ \alpha(Y,X_t)p(Y) \text{ for all x, y}$$

Then the stationary distribution from this kernel corresponds to the target distribution. For a given proposal distribution q and a target distribution p, we need to find $\alpha(X_t, Y)$ and $\alpha(Y,X_t)$ such that the equation holds.

We can actually show the equation holds in three possible cases for any particular X_t, Y pair (1,7)

Case (1): $q(X_t,Y)\ p(X_t)= q(Y,X_t)\ p(Y)$, hence, accordingly set $\alpha(X_t, Y)= \alpha(Y,X_t)=1$ implies the equation holds.

Case (2): $q(X_t,Y)\ p(X_t)> q(Y,X_t)\ p(Y)$, in which case, we have to set

$\alpha(X_t, Y)= \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)}$ and $\alpha(Y,X_t)=1$. It then follows

$$q(X_t,Y)\ \alpha(X_t, Y)p(X_t)= q(X_t,Y)\ \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)}\ p(X_t)$$

$$= q(Y,X_t)\ p(Y)$$

$$= q(Y,X_t)\ \alpha(Y,X_t)p(Y)$$

Case (3): $q(X_t,Y)\ p(X_t)< q(Y,X_t)\ p(Y)$, here we need to have

$\alpha(X_t, Y)=1$ and $\alpha(Y,X_t)= \frac{p(X_t)q(X_t,Y)}{p(Y)q(Y,X_t)}$

Therefore

$$q(X_t,Y)\ \alpha(X_t, Y)p(X_t)= q(X_t,Y)\ \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)}\ p(X_t)$$

$$= q(Y,X_t)\ p(Y)$$

$$= q(Y,X_t)\ \alpha(Y,X_t)p(Y)$$

4.2.2 Proposal distribution for Metropolis-Hasting Algorithm

A good proposal distribution can greatly enhance the performance of this Metropolis-Hasting algorithm. A great proposal distribution produces candidate values that cover the support of the stationary distribution in a reasonable number of iterations (usually in thousands) and meanwhile, the candidate values are not rejected or accepted too frequently (4). Both factors are related to the spread of the proposal distribution. If the proposal distribution chosen is too diffuse, the candidate values will be rejected quite often and will require many more iterations to adequately explore the space of the target distribution. If the proposal distribution is too focused (for example with small variance), the chain might remain in one region of the target distribution for many iterations, and can't cover the other regions of the target distribution.

4.2.3 Independence Chain and Random Walk Chain

In theory, we could choose any function satisfying the conditions above to be our transition kernel $q(X_i,.)$. However, for practical purpose, we need to use one that we can draw candidate values from. Hence, typical choices for $q(X_i,.)$ are uniform, beta, gamma, normal, and other standard distributions (11). Based on different choices of q, we can construct 2 different variants of M-H algorithm: independence chain and the random walk chain.

If the proposal density q is independent of the current value in the sequence, that is p(X,Y)=p(y), then the resulting algorithm is an independence chain. Regardless of what current value it is, the next value is always drawn from the same distribution q. In terms of matrix, we can think of a transition matrix where all rows are the same.

For independence chain, the acceptance probability becomes

$$\alpha(X_t, Y)=\min(1, \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)})=\min(1, \frac{p(Y)q(X_t)}{p(X_t)q(Y)})$$

Therefore, we wish to choose a proposal density q(.) that approximates p(.) very well, which in turn will suggest that a high acceptance probability is desirable. However, although we would like to have q(.) to resemble p(.), the tail behavior of p(.) is more important (4). In particular, if p/q is bounded, the convergence of the Markov Chain to its stationary distribution is faster overall (12). Thus, it is wiser to choose a proposal distribution that is somehow more diffuse than p(.).

Often it is unclear how to determine a decent approximation to the target distribution p(.), and thus one wants a way to generate candidate values that has a great chance of being accepted but will still be able to move through the support of p(.).Random walk chain provides a great alternative. The proposal density is defined by letting the density have the form $q(X_t,Y)=h(Y-X_t)$, where h is a symmetric density about the origin. So we have $q(X_t,Y)=h(Y-X_t)=h(X_t-Y)=q(Y,X_t)$. Because of the symmetry, the acceptance probability is then:

$$\alpha(X_t, Y)=\min(1, \frac{p(Y)q(Y,X_t)}{p(X_t)q(X_t,Y)})=\min(1, \frac{p(Y)}{p(X_t)})$$

Unlike the independence chain, the random walk chain generates the candidate value from a distribution depending on the current value X_t. Usually people think of the random walk chain as the candidate value Y in the current state of the chain X_t plus a $N(0,\sigma^2)$ noise added(13). Other symmetric distribution might also be used instead of a normal, but normal distributions are relatively simple to generate from and provide reasonable results. For random walk chain with normal proposal densities, it has been suggested that acceptance rates between 25% and 45% are good (14, 15). The "best" choice of acceptance rate ranges from 45% for one and two parameters to 25% for problems with more parameters (14, 15).

For random walk chain you have to specify the variance σ^2. We want to choose σ^2 so that the lag 1 autocorrelation ρ_1 is small. If you choose σ^2 to be large, intuitively, we will have candidate values rejected often because a lot of the candidate values are likely to fall out of the likely range of p(.). It will generate a chain with a lot of repeats (the candidate value is rejected, therefore, $X_{t+1}=X_t$) and a large first-order autocorrelation ρ_1. If chosen σ^2 is very small, although we avoid repeats, it might generate a chain that is trapped in certain region and move really slowly over the entire support of p(.) (13). In general, we can try a σ^2 value, and if it doesn't give reasonable result, we can always adjust accordingly: if the chain moves too slowly, increase σ^2; if too many repeats, decrease σ^2 (13).

Overall, a chain is said to be poor mixing if it stays in small regions of the parameter space for long periods of time, as opposed to well mixing china that seems to efficiently explore the whole space (10). We can solve the problem by either changing a starting value or tuning the proposal distribution. It is suggested to use a starting value as close as possible to the center of the target distribution, for easier computation, people always use the mode of a distribution, which can be obtained in R by laplace function. To tune the proposal density, we can adjust mixing, in particular, the acceptance probability. This is generally done by adjusting the standard deviation (SD) or the eigenvalue of the covariance matrix for normal or multi-normal distribution,

increasing or decreasing the range (-a,a) for a uniform proposal density, or changing the degree of freedom if a χ^2 proposal is used (7).

4.3 Gibbs Sampler

4.3.1 Basic Gibbs Sampler

The Gibbs sampler is a special case of the general Metropolis-Hasting Algorithm, specifically adapted for multidimensional target distribution. The idea behind Gibbs sampler is to simplify a multivariate problem into a sequence of conditional univariate problems, and then through MCMC iteration to obtain realizations from the target distribution.

Gibbs sampler is introduced in the context of imaging processing by Geman and Geman in 1984. It considers the univariate conditional distribution, where all of the random variables but one are assigned fixed values. Such conditional distribution is much easier to simulate than complex joint distributions and usually has simple forms (such as normal, inverse χ^2, etc). With Gibbs Sampler, one simulates n random variables sequentially from the n univariate conditional distributions rather than simulating a single n-dimensional vector in a single pass using the full joint distribution (10). By simulating large enough samples, the marginal population characteristics such as mean, variance or any function of p(.), can be obtained with desired accuracy. Notice that in Gibbs sampler, the acceptance probability is always 1, that every sample is accepted.

Let $X=(X_1,X_2,X_3,\ldots X_d)$ be a random vector in R^d, and denote X_{-i} be the d-1 dimensional random vector as

$X_{-i}=(X_1,X_2,\ldots X_{i-1},X_{i+1},\ldots.X_d)$

The conditional distribution of X_i given $X_{-i}=x_{-i}$ is $p(X_i|x_{-i})$.The Gibbs sampling procedure is proceeded as follows (3,4,10,13):

1.Initilize with a starting value $x_{(0)}$ at time t=0.

2.For each iteration, indexed t=1, 2, … repeat:

(a)Set $x_1=X_1(t-1)$

(b)For each coordinate i=1,2,3,….d, generate $X_i^*(t)$ from $p(X_i|x_{-i})$. And update $x_i=X_i^*(t)$

(c)Set $X(t)=(X_1^*(t), X_2^*(t),\ldots.. X_d^*(t))$, since every draw is accepted.

(d) Increment t.

A completion of step (2) for all elements in X within one iteration is called a cycle. It is standard practice to use the latest values for each element in X, rather than conditioning on $X^{(t)}=x^{(t)}$ till a completion of a whole cycle. For example, we use a two-variable case. Suppose we have a pair of random variables (X, Y), then Gibbs Sampler generates sequence of (X,Y) pairs by sampling from p(X|Y) and p(Y|X) given that both conditional distributions are known. First, we specify an initial value $Y_0'=y_0'$. Then, by alternatively generating from

$X_i' \sim p(x| Y_i'=y_i')$

$Y_{i+1}' \sim p(y_{i+1}'|X_i'=x_i')$,

We actually get a sequence of "Gibbs Sampler" of random variables

$Y_0^{'}, X_0^{'}, Y_1^{'}, X_1^{'}, Y_2^{'}, X_2^{'}, Y_3^{'}, X_3^{'}, \ldots Y_t^{'}, X_t^{'}$.

Under relatively general conditions (8), the distribution of $X_i^{'}$ converges to the marginal distribution of X, and so does the distribution of $Y_i^{'}$ (16).

4.3.2 Implementation

All the MCMC methods described above have the correct limiting stationary distribution. The reliability of estimate from a MCMC analysis depends on the extent to which sample averages computed using realizations of the chain compared to their expectation under the limiting stationary distribution of the chain (10). A key issue in the successful implementation of Metropolis-Hasting or any other MCMC sampler is the number of runs (steps) when the chain has run sufficiently long so that it is reasonable to believe that the output adequately represents the target distribution and the can be used for estimation. Typically, the first 1000-5000 draws are discarded (10), which is also referred as the burn-in phase of sampling, and then we can perform various convergence tests to assess whether stationary has indeed been reached.

4.4 Convergence Diagnosis

In practice, we have to monitor the performance of an MCMC algorithm, usually by the value of the acceptance rate, constructing graphs, and computing certain diagnostic statistics on the sequence of simulated draws.

One issue is to detect the burn-in period, where the initial value wears off and the distribution of the simulated draws approach the stationary/target distribution after certain number of iteration. Graphic tools are always helpful. We can examine a time series trace plots of simulated values of the random variables against iteration number. It is particularly important when the starting value is not properly chose to close to be the center of the target distribution, instead, is far away from the center. Nevertheless, we must be cautious that the actual burn-in time may be longer than what's suggested by the trace.

A second concern for MCMC algorithm is the autocorrelation between sampled values. In both Metropolis-Hasting algorithm and Gibbs Sampler, the X_t value drawn at time t is dependent on the value at (t-1). A high correlation indicates that the two successive values provide only a little bit more information about the marginal distribution than a single simulated draw. We can quantify the correlation by using an autocorrelation function. Consider a sequence $\{X_i^t\}$, the kth order autocorrelation ρ_k measures the correlation between X_i^t and X_i^{t+k}, where k is the lag or number of iterations separating the two values. When X is a single parameter, ρ_k can be estimated by

$$\hat{\rho}_k = \frac{\operatorname{cov}(X_t, X_{t+k},)}{\operatorname{var}(X_{t,})} = \frac{\sum_{t=1}^{n-k}(X_t - \bar{X})(X_{t+k} - \bar{X})}{\sum_{t=1}^{n-k}(X_t - \bar{X})\text{^}2}, \text{ with } \bar{X} = \frac{1}{n}\sum_1^n X_t$$

A standard graph is to plot the autocorrelation coefficient against the lag k. If the chain is mixed well, we expect it to be highly correlated with small lags, but the autocorrelation should decay as the lag k increases. Also there is another ratio $\sqrt{(1+\rho)/(1-\rho)}$, called the sample size inflation factor (SSIF), where ρ is the autocorrelation coefficient of the lag of interest (10). Roughly, as SSIF increase, you need $\sqrt{(1+\rho)/(1-\rho)}$ times more values to achieve the same precision as with an uncorrelated sequence of values (10).

A partial autocorrelation plots against the lag is also helpful. The kth partial autocorrelation is the excess correlation not accounted for by a k-1 order autoregressive model (AR_{k-1}). Therefore, if a

first order AR_2 model fits, then the partial autocorrelation of lag three is zero, as the lagged autocorrelation is entirely accounted for by the second-order lag. Both plots may indicate underlying correlation structures that maybe hidden in trace plots.

Another issue arises when one wants to use the simulated draws from MCMC algorithm to compute statistics of interest, such as standard error. Because in MCMC algorithm, we don' have independent samples, one cannot use the standard methods to compute standard errors. Instead, we might use a batch mean methods (5). We divide the sequence $\{X_i^t\}$ into b batches, with equal size v. In each batch, we compute a sample, hence, we will have $\overline{X}_i^1$, $\overline{X}_i^2$ $\overline{X}_i^3$,… $\overline{X}_i^b$. If the lag 1 autocorrelation is small, we can then approximate the standard error of the estimate $\overline{X}_i$by the standard deviation of the batch means divided by the square root of the number of batches: $SE(\overline{X}_i)=SD(\overline{X}_i)/\sqrt{d}$ (5).

There are also formal tests available to test for stationary of the sampler after a given point. One test is the Gelman-Rubin method (14, 17), which compares the behavior of several generated chains with respect to the variance of one or more scalar summary statistics. Estimate of the variance of this statistics is analogous to one-way ANOVA using between-sample and within-sample mean square errors. There is also the Geweke test (18), which splits samples by quartiles (the first 10% and the last 20% for example), after removing the burn-in period draws. If the chain reaches stationary, the means of the two samples should be equal. A modified z-test is used to compare the two samples and generates a statistics often referred as the Geweke z-score. A value larger than 2 usually indicates that the means are different, stationary has not yet been reached, and more iterations.

4.5 Examples using Metropolis-Hasting Algorithm and Gibbs Samper

4.5.1 Simple random walk chain

We illustrate the R code for the M-H algorithm with a simple random walk example: the target distribution is a normal (0,1), the proposal distribution is a symmetric unif(0,1).

```
# metropolis for N(0,1) based on uniform candidates

# function

metrop<-function (n, alpha)

+ {

+       vec=vector("numeric", n)

+       x=0

+       vec[1]=x

+       for (i in 2:n) {

+             can=x+runif(1, -alpha, alpha)

+             aprob=dnorm(can)/dnorm(x)

+             u=runif(1)

+             if (u < aprob)
```

```
+                    x=can
+               vec[i]=x
+        }
+        vec
+ }
# example
normvec=metrop(10000,1)
op=par(mfrow=c(2,1))
plot(ts(normvec))
hist(normvec,30)
par(op)
```

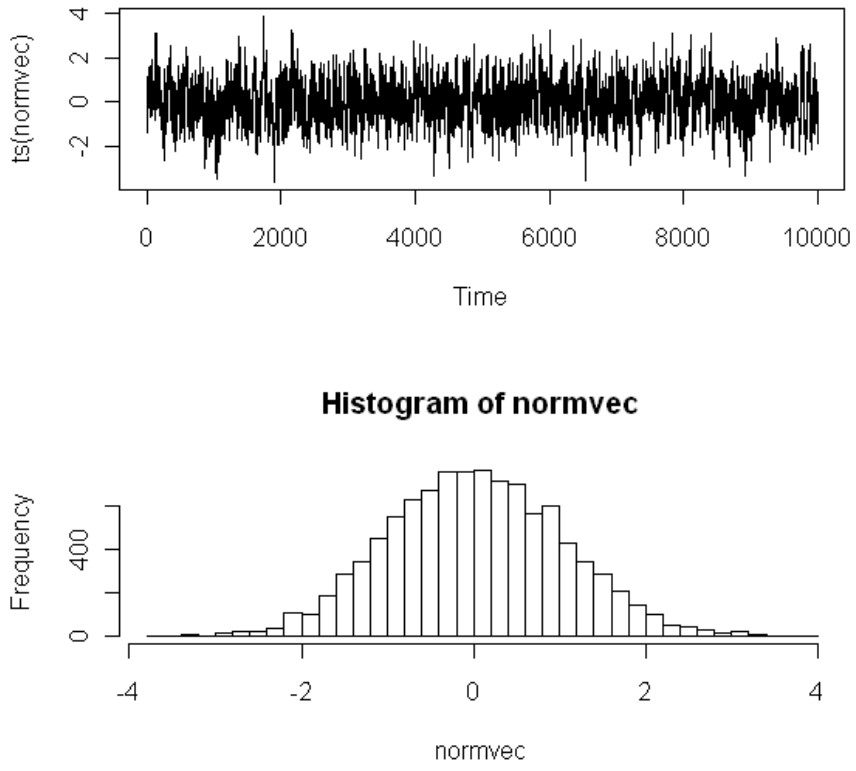


Figure8 A simple random walk chain for normal distribution using a symmetric uniform proposal distribution. (a) Trace plot of the simulated values versus iteration; (b) Histogram of simulated values.

We can see that the trace plot indicates the simulation works pretty well, confirmed with the histogram of the normal bell shape.

4.5.2 Gibbs Sampler example

Next, we will illustrate the R code for Gibbs Sampling to generate a bivariate normal distribution with mean vector (μ_1, μ_2), variance (σ_1^2,σ_2^2) and correlation coefficient ρ.

In the bivariate case, $X=(X_1, X_2)$. The conditional distribution is univariate normal with parameters:

$E[X_2|x_1]= \mu_2+\rho\ \sigma_2/\ \sigma_1(x_1- \mu_1)$,

$Var[X_2|x_1]=(1-\rho^2)\ \sigma_2^2$,

And the chain is generated by sampling from:

$p(x_1|x_2)\sim$Normal $(\mu_1+\rho\ \sigma_1/\ \sigma_2(x_2- \mu_2), (1-\rho^2)\ \sigma_1^2)$

$p(x_2|x_1)\sim$Normal $(\mu_2+\rho\ \sigma_2/\ \sigma_1(x_1- \mu_1), (1-\rho^2)\ \sigma_2^2)$

```
#initialize constans and parameters

N=5000

burn=1000

X=matrix(0,N,2)

rho=0.75

mu1=0

mu2=2

sigma1=1

sigma2=.5

s1=sqrt(1-rho^2)*sigma1

s2=sqrt(1-rho^2)*sigma2

#generate the chain

X[1,]=c(mu1,mu2)

for (i in 2:N){

+  x2=X[i-1,2]

+  m1=mu1+rho*(x2-mu2)*sigma1/sigma2

+  X[i,1]=rnorm(1,m1,s1)

+  x1=X[i,1]
```

```
+   m2=mu2+rho*(x1-mu1)*sigma2/sigma1
+   X[i,2]=rnorm(1,m2,s2)
+ }
b=burn+1
x=X[b:N,]
#cmopare sample statistics to parameters
colMeans(x)
[1] -0.01436258  1.99468353
cov(x)
          [,1]      [,2]
[1,] 1.0083493 0.3779433
[2,] 0.3779433 0.2565535
cor(x)
          [,1]      [,2]
[1,] 1.0000000 0.7430742
[2,] 0.7430742 1.0000000
```

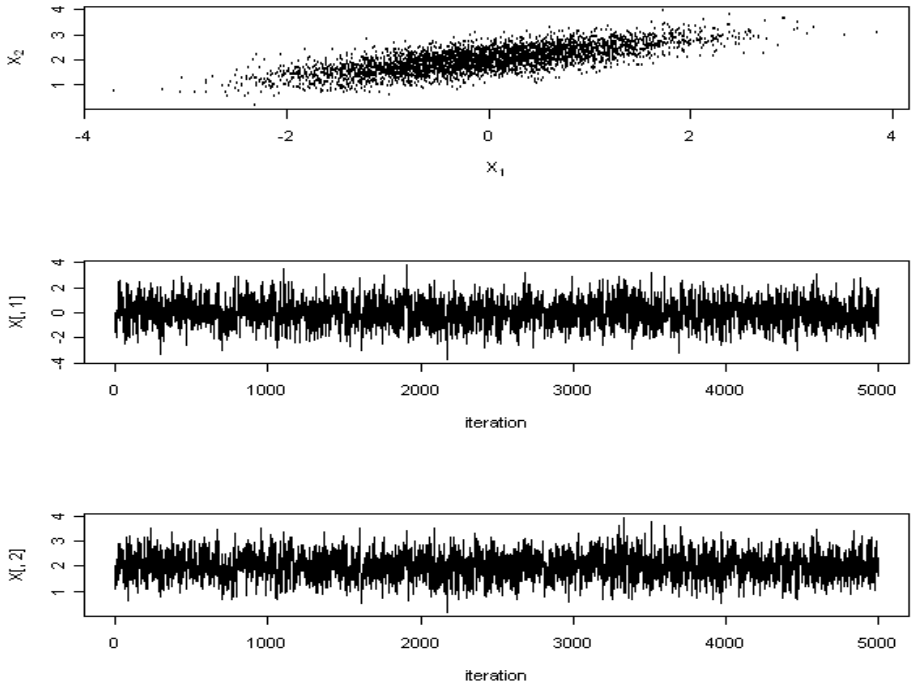


Figure9 (a) Bivariate normal chain generated by the Gibbs Sampler; (b) Trace plot of one variable versus iteration; (c) Trace plot of the second variable versus iteration.

4.5.3 Metropolis-Hasting sampler

With the same example, we explore it using a Metropolis-Hasting Algorithm to simulate a bivariate normal distribution samples using a bivariate normal proposal distribution.

```
#Metropolis-Hasting algorithm

#using MASS package to simulate multivariate normal random variables

#definte a bivariate normal function to evalue the density of simulated draws

dmvnorm=function(x,mu,Sigma){

+ x1=x[1]

+ x2=x[2]

+ mu1=mu[1]

+ mu2=mu[2]

+ sigma1=sqrt(Sigma[1,1])

+ sigma2=sqrt(Sigma[2,2])

+ rho=sqrt(Sigma[1,2]/(sigma1*sigma2))

+ part1=1/(2*pi*sigma1*sigma2*rho)

+ part2=(x1-mu1)^2/sigma1^2+(x2-mu2)^2/sigma2^2-2*rho*x1*x2/(sigma1*sigma2)

+ part3=-1/2*(1-rho^2)

+ part4=exp(part2*part3)

+ total=part1*part4

+ return(total)}


#initialize the values

N=5000

burn=1000

mu=c(0,2)

Sigma=matrix(c(1,0.75,0.75,1),nrow=2)
```

```
u=runif(N)
X=matrix(0,N,2)
k=0 #count the number of acceptance

#generate the randow walk chain using bivariate normal as a jumping distribution
X[1,]=c(1,1)
for (i in 2:N){
+   xt=X[i-1,]
+   y= mvrnorm(1,X[i-1,],.2*Sigma)
+   ratio=dmvnorm(y,mu,Sigma)/dmvnorm(xt,mu,Sigma)
+ if(u[i]<=ratio)X[i,]=y
+   else{
+   X[i,]=xt
+   k=k+1  #y is rejected
+   }
+ }
print(k)
[1] 322
#initialize the values
N=5000
burn=1000
mu=c(0,2)
Sigma=matrix(c(1,0.75,0.75,1),nrow=2)
u=runif(N)
X=matrix(0,N,2)
k=0 #count the number of acceptance

#generate the randow walk chain using bivariate normal as a jumping distribution
X[1,]=c(1,1)
```

```
for (i in 2:N){
+   xt=X[i-1,]
+   y= mvrnorm(1,X[i-1,],.2*Sigma)
+   ratio=dmvnorm(y,mu,Sigma)/dmvnorm(xt,mu,Sigma)
+ if(u[i]<=ratio)X[i,]=y
+   else{
+   X[i,]=xt
+   k=k+1  #y is rejected
+   }
+ }
print(k)
[1] 331
b=burn+1
x=X[b:N,]

#cmopare sample statistics to parameters
colMeans(x)
[1] 5.464745 6.717853
cov(x)
         [,1]     [,2]
[1,] 17.24710 14.68524
[2,] 14.68524 16.10575
cor(x)
         [,1]     [,2]
[1,] 1.000000 0.881115
[2,] 0.881115 1.000000
```

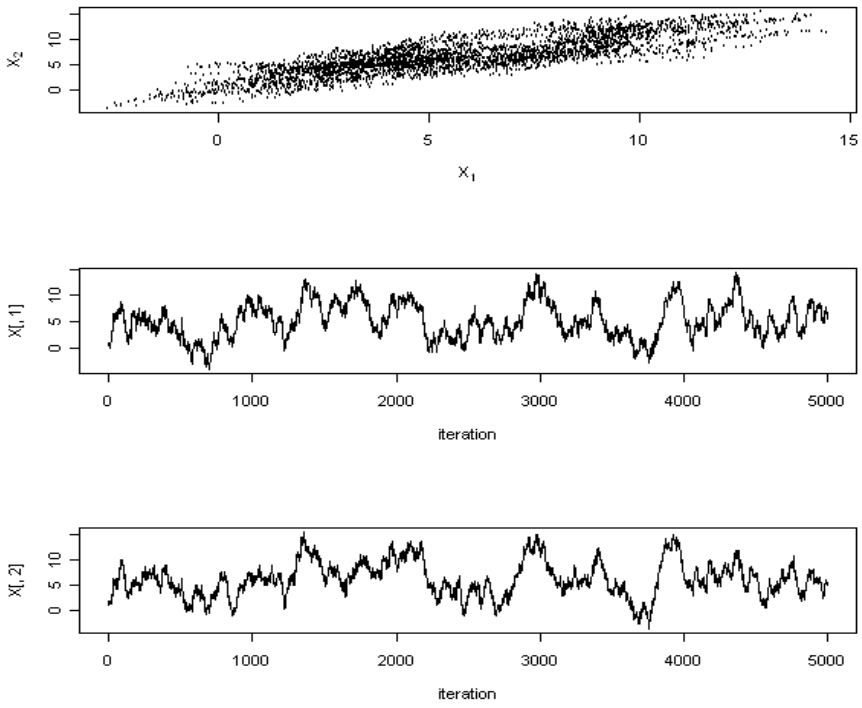


Figure10 (a) Bivariate normal chain generated by the Metropolis-Hasting Sampler; (b) Trace plot of one variable versus iteration; (c) Trace plot of the second variable versus iteration.

For illustration purpose, the proposal distribution for this Metropolis-Hasting algorithm is also bivariate normal distribution with a different variance (0.2 times of the target distribution) intentionally. We actually compare our simulated samples from Gibbs Sampler and Metropolis-Hasting Algorithm (random walk chain) with the target distribution, the variance-covariance matrix from Gibbs sampler is closer to the target than that from Metropolis-Hasting Algorithm. Also from the trace plots, Gibbs Sampler is smoother compared with Metropolis-Hasting which has some repeats. The acceptance rate of the M-H is only 331/5000, about 7%, indicating poor mixing. To improve the chain, we can tune the variance-covariance matrix for the proposal density to increase the acceptance rate.

Chapter 5 Summary and Conclusion

Gibbs Sampler is the simplest algorithm to simulate a Markov Chain. It shows advantage as the dimension of a problem increases. Also, it is always a good choice for conditional conjugate models, such as the example above, where, we can directly sample from each conditional distribution.

Metropolis-Hasting algorithm, on the other hand, seems require a little extra. We have to find a decent proper proposal distribution to perform this algorithm. Then we need to tune the variance-covariance matrix (for random walk chain) to get better mixing and to improve the acceptance rate.

In situation of non-conjugate conditional distribution, or where the conditional distribution is not so easy to directly sample from, we can't use Gibbs Sampler directly, instead, we would use Metropolis-Hasting embedded within Gibbs Sampler to update parameters alternatively. This is called "Metropolis within Gibbs".

The ultimate value of Gibbs Sampler lies in its practical potentials. With the groundwork been laid in the pioneering papers of Geman and Geman (19), Tanner and Wong (20) and Gelfand and Smith (21), research using the Gibbs sampler is exploding. The application includes DNA and protein sequencing, logistic regression model for cancer classification and prediction (22), time series analysis (23), automobile product market analysis (24).

Innovations in theory and practice of MCMC continue at a rapid pace. Metropolis-Hasting Algorithm and the Gibbs Sampler provide building blocks for more advanced algorithm.

Appendix

R codes for Figure 1

```
#  Discrete  prior
p = seq(0.05, 0.95, by = 0.10)
prior = c(1, 2, 2, 3, 7, 7, 4, 3, 1, 1)
prior = prior/sum(prior)
plot(p, prior,type = "h", ylab="Prior Distribution")
data=c(7,3)
post=pdisc(p, prior, data)
cbind(p, prior, post)
plot(p, post, type = "h", ylab="Posterior Distribution")
```

R codes For Figure 3

```
#conjugate prior
p= seq(0, 1, length = 100)
a = 5
b = 5
s = 7
f = 3
prior = dbeta(p, a, b)
like = dbeta(p, s+1, f+1)
post=dbeta(p, a+s,b+f)
curve(dbeta(x,a+s,b+f), from=0, to=1,
xlab="p",ylab="Density",lty=1,lwd=4)
curve(dbeta(x,s+1,f+1),add=TRUE,lty=2,lwd=4)
curve(dbeta(x,a,b),add=TRUE,lty=3,lwd=4)
legend(.7,4,c("Prior","Likelihood","Posterior"),
lty=c(3,2,1),lwd=c(3,3,3))
```

R codes For Figure 4

```
#  HISTOGRAM PRIOR
midpt = seq(0.05, 0.95, by = 0.1)
prior = c(1, 2, 2, 3, 7, 7, 4, 3, 1, 1)
prior = prior/sum(prior)
curve(histprior(x,midpt,prior), from=0, to=1, ylab="Prior density",ylim=c(0,.3))
# The curve command chooses an appropriate number of values
# to form a grid between the end points and computes the
# value of the posterior distribution at each point of that grid.
s = 7
f = 3
curve(histprior(x,midpt,prior)*dbeta(x,s+1,f+1),  from=0, to=1, ylab="Posterior density")
x=seq(0,1,length=500)
post=histprior(x,midpt,prior) * dbeta(x,s+1,f+1)
post=post/sum(post)
ps=sample(x, replace=TRUE,prob=post)
hist(ps)
```

R codes For Figure 5 and Figure 6

```
#rejection sampling from a beta distribution: x~beta(3,4)
betapdf=function(x){
60*(x^3)*((1-x)^2)}
curve(betapdf,0,1)
abline(2.1,0)

n <- 1000
k <- 0      #counter for accepted
j <- 0      #iterations
z <- numeric(n)
```

```
while (k < n) {
u <- runif(1)
j <- j + 1
y <- runif(1)  #random variate from g
if (60/(2.1)*(y^3) * ((1-y)^2) > u) {
#we accept y
k <- k + 1
z[k] <- y
 }
}
j
```

R codes For Figure 7

```
# Monte Carlo estimation of ∫_0^1 e^(-x)/(1+x^2) dx with two different g(x):g1(x)=e-x and g2(x)=e-x/(1-e-1)
n <- 10000
theta.hat <- variance<- st.dev <- numeric(2)
g <- function(x) {
exp(-x)/(1+x^2) * (x > 0) * (x < 1)
}

#1 st candidate of g(x)
x<-rexp(n)
f<-g(x)/(exp(-x))
theta.hat[1]=mean(f)
theta.hat[1]

variance[1]=var(f)
variance[1]
```

```
st.dev[1]=sd(f)
st.dev[1]

#2nd candidate of g(x)
u<-runif(n)
x<-log(1/(1-u*(1-exp(-1))))
f<-g(x)*exp(x)*(1-exp(-1))
theta.hat[2]=mean(f)
theta.hat[2]

variance[2]=var(f)
variance[2]
st.dev[2]=sd(f)
st.dev[2]

rbind(theta.hat, st.dev)
```

R codes For Figure 8

```
# 4.5.1 a simple random walk example: the target distribution is a normal (0,1),
#the proposal distribution is a symmetric unif(0,1)
metrop<-function (n, alpha)
{
vec=vector("numeric", n)
x=0
vec[1]=x
for (i in 2:n) {
can=x+runif(1, -alpha, alpha)
aprob=dnorm(can)/dnorm(x)
u=runif(1)
```

```
if (u < aprob)
x=can
vec[i]=x
}
vec
}

# example
normvec=metrop(10000,1)
op=par(mfrow=c(2,1))
plot(ts(normvec))
hist(normvec,30)
par(op)
```

R codes for Figure 9

```
#4.5.2 Gibbs Sampler
#initialize constans and parameters
N=5000
burn=1000
X=matrix(0,N,2)
rho=0.75
mu1=0
mu2=2
sigma1=1
sigma2=.5
s1=sqrt(1-rho^2)*sigma1
s2=sqrt(1-rho^2)*sigma2
#generate the chain
```

```
X[1,]=c(mu1,mu2)
for (i in 2:N){
x2=X[i-1,2]
m1=mu1+rho*(x2-mu2)*sigma1/sigma2
X[i,1]=rnorm(1,m1,s1)
x1=X[i,1]
m2=mu2+rho*(x1-mu1)*sigma2/sigma1
X[i,2]=rnorm(1,m2,s2)
}
b=burn+1
x=X[b:N,]
#compare sample statistics to parameters
colMeans(x)
cov(x)
cor(x)

par(mfrow=c(3,1))
plot(x,main="",cex=0.5,xlab=bquote(X[1]),ylab=bquote(X[2]),ylim=range(x[,2]))
plot(X[,1],type="l",xlab="iteration")
plot(X[,2],type="l",xlab="iteration")
```

R codes for Figure 10

```
#4.5.3 Metropolis-Hasting algorithm
#using MASS package to simulate multivariate normal random variables
#definte a bivariate normal function to evalue the density of simulated draws
dmvnorm=function(x,mu,Sigma){
x1=x[1]
x2=x[2]
mu1=mu[1]
```

```
mu2=mu[2]
sigma1=sqrt(Sigma[1,1])
sigma2=sqrt(Sigma[2,2])
rho=sqrt(Sigma[1,2]/(sigma1*sigma2))
part1=1/(2*pi*sigma1*sigma2*rho)
part2=(x1-mu1)^2/sigma1^2+(x2-mu2)^2/sigma2^2-2*rho*x1*x2/(sigma1*sigma2)
part3=-1/2*(1-rho^2)
part4=exp(part2*part3)
total=part1*part4
return(total)}

#initialize the values
N=5000
burn=1000
mu=c(0,2)
Sigma=matrix(c(1,0.75,0.75,1),nrow=2)
u=runif(N)
X=matrix(0,N,2)
k=0 #count the number of acceptance

#generate the random walk chain using bivariate normal as a jumping distribution
X[1,]=c(1,1)
for (i in 2:N){
xt=X[i-1,]
y= mvrnorm(1,X[i-1,],.2*Sigma)
ratio=dmvnorm(y,mu,Sigma)/dmvnorm(xt,mu,Sigma)
if(u[i]<=ratio) X[i,]=y
else{
X[i,]=xt
```

```
k=k+1  #y is rejected
 }
}
print(k)

b=burn+1
x=X[b:N,]
#compare sample statistics to parameters
colMeans(x)
cov(x)

par(mfrow=c(3,1))
plot(x,main="",cex=0.5,xlab=bquote(X[1]),ylab=bquote(X[2]),ylim=range(x[,2]))
plot(X[,1],type="l",xlab="iteration")
plot(X[,2],type="l",xlab="iteration")
```

Bibliography

1. Chib S, Greenberg E, Understanding the Metropolis-Hasting Algorithm, The American Statistician, Vol 49, No.4, 327-335, 1995

2.Gelman A, Carlin J.B, Stern H.S, Rubin D.B, Bayesian Data Analysis, Chapman &Hall, 2004

3. Rizzo M.L, Statistical Computing with R, Chapman &Hall,2008

4. Givens G.H, Hoeting J.A, Computational Statistics, Wiley ,2005.

5. Albert Jim, Bayesian Computation with R, Springer, 2007.

6. Rosenbluth M.N, Teller A.H and Teller E, "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics , 1953.

7. Hasting W.K, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications", Biometrika 57 (1):97-109, 1970.

8. Roberts G.O and Polson N.G, "On the Geometric Convergence of the Gibbs Sampler", J.R.Statistics, 56, No.2: 377-384, 1994.

9. Jeffery H, Theory of Probability. Oxford University Press, New York, 3rd edition, 1961

10.Walsh C.B, Markov Chain Monte Carlo and Gibbs Sampling, Massachusetts Institute of Technology press, EEB581 Lecture notes, 2004.

11.Gelman A, Roberts G.O, Gilks W.R, Efficient Metropolis Jumping Rules, Bayesian Statistics, Oxford University Press, 599-507,1996,.

12. Roberts C.P, Casella G, Monte Carlo Statistical Methods. Springer-Verlag, New York, 1999

13. Viele K, Markov Chain Monte Carlo, STA705 Lecture notes, 2006.

14. Gelman A, Rubin D,B, Inference From Iterative Simulation Using Multiple Sequences, Statistical Science, 7:457-511,1992.

15. Roberts G.O, Gelman A, Gilks W.R, Weak Convergence and Optimal Scaling or Random Walk Metropolis Algorithms. The Annals of Probability, 7(1):110-120, 1997.

16.Casella G, George E.I, Explaining the Gibbs Sampler, American Statistician, Vol.46, No.3, 167-174,1992.

17. Gelman A, Inference and Monitoring Convergence, Markov Chain Monte Carlo in Practice, Chapman &Hall, 131-143, 1996

18.Geweke J. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. Bayesian Statistics, 169-193,1992.

19. Geman S, Geman, D, Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images, IEEE Transactions on Pattern Analysis and Machine Intelligence, 6: 721-741, 1984.

20. Tanner M.A., and Wong W, The Calculation of Posterior Distributions by Data Augmentation, Journal of the American Statistical Association, 82:528-550, 1984.

21. Gelfand A.E, Smith A.F.M, Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling, Journal of the American statistical Association, 85, 1990, 972-985.

22. Zhou X, Liu K.Y,Wong S.T, Cancer Classification and Prediction Using logistic Regression with Bayesian Gene Selection. Journal of Biomed Information, Vol.37, 2004, 249-259.

23. Waggoner D.F, Zha T, A Gibbs Sampler for structural Vector Autoregressions, Journal of Economic Dynamics and Control, Vol 28, Nov 2003, 349-366.

24. Romeo C.J, A Gibbs Sampler for Mixed Logit Analysis of Differentiated Product Markets using Aggregate data, Computational Economic, Vol 29, Feb 2007, 33-68.

Vita

Wenwen Sun was born in Hubei province in China on September 22nd, 1981, the daughter of Kefu Sun and Shaohua Ren. After completing her work in high school in 1999, she entered Wuhan University in Wuhan, Hubei province, China. She received the Bachelor Degree of Science in Biology in 2003. Then she went to US and entered the graduate school of University of Texas at Austin to continue pursuing a Master degree in Microbiology. After graduating with a Master degree in Microbiology in 2007, she was accepted as a master student in the department of Statistics in the University of Texas at Austin.

Permanent Address: 70 Greene St, Apt 1603

Jersey City, NJ, 07302

This report was typed by the author.